Computing and Informatics, Vol. 23, 2004, 537–556

BUILT-IN SELF-TEST QUALITY ASSESSMENT USING HARDWARE FAULT EMULATION IN FPGAS

Abílio Parreira, J. Paulo Teixeira, Marcelino B. Santos

IST/INESC-ID
R. Alves Redol, 9
1000-029 Lisboa, Portugal
e-mail: marcelino.santos@inesc-id.pt

Manuscript received 18 February 2005

Abstract. This paper addresses the problem of test quality assessment, namely of BIST solutions, implemented in FPGA and/or in ASIC, through Hardware Fault Emulation (HFE). A novel HFE methodology and tool is proposed, that, using partial reconfiguration, efficiently measures the quality of the BIST solution. The proposed HFE methodology uses Look-Up Tables (LUTs) fault models and is performed using local partial reconfiguration for fault injection on Xilinx^{TM} Virtex and/or Spartan FPGA components, with small binary files. For ASIC cores, HFE is used to validate test vector selection to achieve high fault coverage on the physical structure. The methodology is fully automated. Results on ISCAS benchmarks and on an ARM core show that HFE can be orders of magnitude faster than software fault simulation or fully reconfigurable hardware fault emulation.

Keywords: Hardware Fault Emulation, fault coverage, FPGA, BIST, ASIC

1 INTRODUCTION

High-quality electronic products require high-quality test. Design for Testability (DfT) and test preparation during the design phase are mandatory for cost-effective product development [1]. Test resource partitioning makes the use of Built-In Self-Test (BIST) technology very attractive for production and lifetime testing [2]. In this context, *test quality assessment* is a crucial factor of success.

Test quality can be evaluated through four quality metrics: TE (Test Effectiveness), TO (Test Overhead), TL (Test Length) and TP (Test Power). TE measures

the ability of a given test pattern to uncover likely physical defects induced during IC manufacturing or lifetime operation. TL is the number of test vectors, directly proportional to test application time. TO quantifies the additional hardware resources (test hardware, and I/O pins) and its impact on system performance. TP computes the power consumption needed to perform the test process. Test effectiveness, associated with Defects Coverage [3], is usually evaluated through the FC (Fault Coverage) metrics, using a fault model, such as the single Line Stuck-At (LSA) fault model. In this work, only static faults are considered. Out of the four quality indicators, TE embodies the necessary condition: if the target coverage value is not met, the test solution is insufficient, and must be upgraded. This paper focuses on the costs of TE evaluation, i.e., of fault coverage computation, for System on a Chip (SoC) cores. Traditionally, FC evaluation is performed running efficient software tools – that is, by *software fault simulation* (SFS). However, for complex cores, for large fault lists and/or for large number of test vectors, SFS costs easily become prohibitive.

Hence, the main objective of this paper is to show that *hardware fault emulation* (HFE), using FPGA (Field Programmable Gate Arrays), can provide a cost-effective solution. A novel methodology and tool, using partial reconfiguration for fault injection, is proposed.

A set of characteristics associated with BIST solutions are considered, and addressed in the HFE process proposed in this paper. BIST technology routinely uses PR (pseudo-random) TPG (test pattern generators), like LFSR (Linear Feedback Shift Registers). However, random pattern-resistant faults require refined TPG approaches, e.g., weighted PR, or re-seeding techniques [2]. Still, *long test sequences* must be applied for FC computation. The modules under self-test may be combinational or sequential. Although many efficient algorithms have been proposed for SFS (e.g., [4, 5]), for complex *sequential circuits* SFS is still a very time-consuming task and it can significantly lengthen time-to-market. On the contrary, HFE can speed-up run times orders of magnitude. Another FS task not efficiently performed by software tools is *multiple fault simulation*, mainly due to the huge fault list. However, multiple fault simulation may become mandatory, namely in the product certification process of safety-critical applications [6]. Nevertheless, HFE may efficiently cope with these issues. Finally, the *design style* can influence the FS solution to be selected. In fact, volume production usually determines whether a product is implemented in ASIC (Application Specific Integrated Circuit) or in FPGA. Hence, HFE results usefulness depend on the structural description of the core (FPGA, or ASIC), and on the selected *fault model* used for FC computation. In this work we use the LSA fault model.

The purpose of this paper is threefold. First, it is to present a highly efficient HFE methodology and tool for test quality assessment. The methodology uses a partial reconfiguration technique for fault injection on Xilinx^{TM} Virtex and/or Spartan FPGA components, by deriving very small bit files for fault injection. These bit files are obtained from the full configuration bit file, by means of its direct manipulation, without requiring any additional software tools. With the proposed

HFE methodology, a set of Look-Up-Tables (LUT) based fault models (coarse and fine grain models) is introduced. Second, for an FPGA product, LUT fault injection effectiveness is evaluated as a fault sampling technique. Third, for an ASIC product, HFE test vector selection is evaluated by the ability of the selected vectors to detect hard faults in the target ASIC structure.

This paper is organized as follows: in Section 2, previous work on HFE is briefly reviewed. In Section 3, the proposed HFE methodology is presented, namely the LUT extraction and the partial reconfiguration processes. Section 4 presents the set of LUT-based fault models. Section 5 describes experimental results that evaluate the fault sampling and compare SFS and HFE, for an FPGA design style. Section 6 evaluates HFE capability to select test vectors to test ASIC structures. Finally, Section 7 summarizes the main conclusions of this work.

2 PREVIOUS WORK ON HFE

Different HFE approaches using FPGAs have been proposed in the literature to overcome the difficulties with SFS for sequential circuits. These deal with two key parameters of the fault injection campaign: (1) additional FPGA resource allocation and (2) time to perform the HFE process, tHFE. This time needs to be compared with the time required to perform SFS, i.e., tSFS.

Dynamic fault injection, using dedicated extra hardware, and allowing the injection of different faults without reconfiguring the FPGA, was proposed in [7–10]. The additional hardware proposed for implementing dynamic fault injection uses a shift register whose length corresponds to the size of the fault list. Each fault is injected when the corresponding position in the shift register has logic "1", while all other positions have logic "0". Upon initialization, only the first position of the shift register is set to "1". Then, the "1" is shifted along the shift register, activating one fault at a time. This technique was further optimized in [11]. It is time efficient. However, the added hardware increases with the number of faults to inject, thus limiting the size of the circuits that can be simulated. This is a major limitation of this technique.

In [8], it is shown that parallelism is possible by injecting independent faults at the same time. This parallelism is limited to faults in different propagation cones. However, the reported speedup is only 1.36 times the pure serial fault simulation.

In [12], a new approach that included a backward propagation network to allow critical path tracing [13] is proposed. This information allows multiple faults to be simulated for each test vector. Nevertheless, it also requires significant amount of extra hardware. Only combinational circuits have been analyzed.

A serial FS technique that requires partial reconfiguration during logic emulation was proposed in [14]. This *static fault injection* technique requires *no* extra hardware for fault injection. The authors show that HFE can be two orders of magnitude faster than SFS, for designs over 100 000 gates. Hence, using partial reconfiguration in FPGA, it is possible to significantly accelerate the FS process ($t_{HFE} \ll t_{SFS}$).

More recently, other static fault injection approaches were proposed in [15–17] using the JBITS [18] interface for partial FPGA reconfiguration. The JBITS software can achieve effective injection of faults on LUT [15, 16] or on erroneous register values [17]. These papers do not report any actual results on partial FPGA reconfiguration times for fault injection – only predictions are given. Moreover, the approach that uses JBITS requires the Java SDK 1.2.2 platform [19] and the XHWIF hardware interface [20].

3 PROPOSED HFE METHODOLOGY

In this work, a novel HFE methodology is introduced, taking advantage of the limited reconfiguration effort required to inject faults in LUTs, by an efficient partial reconfiguration technique. First, a description of the LUT extraction and fault injection procedures is presented. Then the proposed LUT-based fault models are described in Section 4. The usefulness of the proposed HFE approach is demonstrated in Section 5 for FPGA products, and in Section 6 for ASIC products. As, in the proposed methodology, no additional FPGA resource allocation is required, the key quality indicator for HFE is tHFE, which is computed as

$$t_{HFE} = t_{conf} + NF.t_{reconf} + [TL/f_{HFE}] \quad (1)$$

where NF is the fault list size, TL the number of test vectors to be applied in the BIST session, f_{HFE} is the clock frequency at which the FPGA is operated during the BIST session and t_{conf}, t_{reconf} represent the configuration and partial reconfiguration times. For a given fault list, t_{HFE} can be limited by reduced t_{reconf} and by the at-speed operation. In fact, for usual clock frequencies, the test length of the BIST session, TL, does not pose a severe constraint (as it does in SFS).

3.1 LUT Extraction

Xilinx Virtex [21] and Spartan FPGA components are used in this work, due to the fact that partial reconfiguration of these components is possible and documented [22]. CLBs (Configurable Logic Blocks) are the building blocks for implementing custom logic. Each CLB contains two slices. Each slice contains two 4-input LUTs, 2 flip-flops and some carry logic. In order to extract the LUT's configuration from the binary file, the software tool developed to implement the methodology analyses the part of the frames[1] that describes the CLBs configuration, as depicted in Figure 1.

Each LUT can be used to implement a logic function of 0, 1, 2, 3 or 4 inputs. For a given application and its allocated resources, the *number of active inputs* for

[1] A frame is the minimum amount of bits that is required when configuring an FPGA. To reconfigure an LUT 16 frames are sent plus a dummy one to flush the pipeline.The length of a frame is related with the number of CLB's rows, and must be word sized (32 bits multiple).

each assigned LUT must be identified. This allows fault list generation, needed to automate the fault injection process.

Since each LUT has 4 inputs, the LUT contents consist of a 16-bit vector, one vector for each input combination. Let y_{abcd} be the output value for the combination of input values abcd. The LUT configuration 16-bit vector can be denoted y_{0000}, y_{0001}, y_{0010}, ..., y_{1111}, where each bit position corresponds to the LUT output value for the respective combination of the inputs i_3, i_2, i_1 and i_0. If one input has a fixed value, then an 8-bit vector is obtained. For instance, if we have always i_3=0, then the relevant 8-bit vector is y_{0000}, y_{0001}, y_{0010}, y_{0011}, y_{0100}, y_{0101}, y_{0110}, y_{0111}. After retrieving the information of each LUT from the bit file, the relevance of each input ix (x = 0, 1, 2 and 3) is evaluated, by comparing the 8-bit vectors corresponding to $i_x = 0$ and $i_x = 1$. If these two vectors are identical, then the input i_x is not active. This allows the identification of each LUT type, e.g., LUT2, LUT3 or LUT4, according to their numbers of relevant inputs. In this LUT extraction process, the nets associated with the LUT inputs and output are also identified, in order to establish a link between the faults to inject and the design.

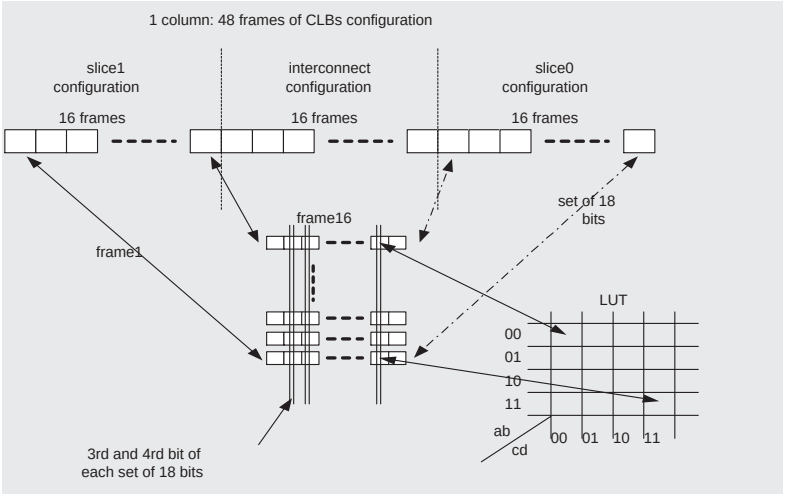

Fig. 1. Positions in the frames of the LUT configuration bits

3.2 Fault Injection

After LUT extraction, fault injection is carried out by partial reconfiguring of the FPGA. The modification of the LUT contents (Karnaugh map) depends on the user's defined fault model. The proposed HFE methodology supports a generic fault model for LUTs that is presented in Section 4.1. Partial reconfiguration bit files, targeting each fault, are automatically generated in sequence. In this way,

each bit file reprograms only the minimum number of frames required to (1) un-inject the previous fault and (2) inject the next one. The binary file for partial reconfiguration requires the faulty frames and a more restricted number of commands [27].

This partial reconfiguration is repeated during the fault simulation (FS) process. At present, this process is automated only for BIST. However, other HFE strategies may be considered, namely emulation of the FS process on a target device. In order to perform FS, the developed tool sends a [start BIST] command and waits for [BIST end] to read the MISR signature. At present, this interaction between the software that controls the fault simulation process and the FPGA is accomplished using the JTAG 200 Kb parallel III port. The HFE control is made using the USER0 and USER1 instructions of the TAP controller implemented in the FPGA: USER0 selects the control mode and USER1 selects the data transfer mode. As shown is Figure 2, fault injection is performed by partial reconfiguration of the FPGA. The start test is performed by sending a start bit over the data pin after the USER0 instruction has been issued. The end of the BIST session is sensed by reading a bit over the data pin after the USER0 instruction has been issued. The BIST signature is obtained by reading bits over the data pin after the USER 1 instruction has been issued.

The validation of the tool is carried out in a *DIGILAB 200 E* board with a Xilinx Spartan 200 E FPGA. However, the developed software tool supports any board with JTAG interface with Virtex or Spartan FPGAs, including the E type.

The developed tool delivers a fault simulation report. This report includes, for each fault, the target LUT type and location, equivalent faults, detection result and MISR signature. The LUT inputs and output are also identified, using the information from the LUT extraction process.

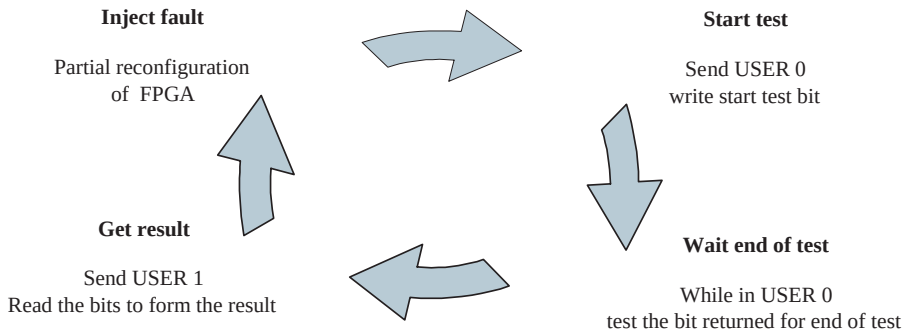

Fig. 2. HFE control states

4 FAULT MODELS

4.1 LUT Fault Models

In order to test a circuit implemented in an FPGA, ensuring that its functionality is correct with a certain degree of confidence (measured by test effectiveness, TE), not only structural faults must be modeled, but also the FPGA configuration must be tested.

The most efficient HFE approaches are based in LUT fault injection. In fact, it is possible to inject/un-inject LUT faults very efficiently. Moreover, for a given application and its correspondent resource allocation, the LUTs are used to implement a significant part of the circuit's logic. As a consequence, when using LUT faults to estimate the fault coverage of the entire fault set, the designer uses a significant sample of the complete fault list. This is demonstrated in Section 5.

LSA faults are the type of faults most commonly modeled in LUTs. Hence, the *coarse LUT fault model* used in the proposed methodology is the I/O LSA fault model. The reconfiguration vector that corresponds to e.g. the LUT input A stuck at value v is obtained by copying the values y_{vBCD} to $y_{\overline{v}BCD}$ for each BCD combination. For instance, the vector for the fault input A LSA-1 is obtained, as illustrated in Figure 3, by copying y_{1000} to y_{0000}, y_{1001} to y_{0001}, y_{1110} to y_{0110}, ..., y_{1111} to y_{0111}. LUT fault collapsing can be made easily by identifying the faults that require the same faulty LUT content.

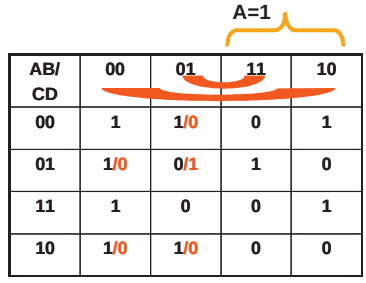

Fig. 3. Computing the 16 bits for LUT injection of an "input A LSA-1" fault

In order to increase the efficiency of the HFE process, the faulty LUT content is pre-computed for each possible fault in the fault-free LUT configuration. The LUT is described as a 16-bit vector. Different fault models may modify the fault-free Karnaugh map in different ways. Hence, a set of LUT fault models can be users defined. The definition of each fault model is made in a file that associates one or several LUTs faulty contents to each possible combination of the 16 bits that correspond to the fault free LUT configuration. The simulation tool, after loading this file, can easily inject all the faults associated to each LUT configuration, previously obtained as described in Section 3.2.

An auxiliary tool was developed to generate the faulty LUT configurations for different fault models. It is a major advantage to perform LUT faulty configuration calculation only once and prior to fault simulation. In fact, this is more efficient and adds flexibility to the fault model.

Following the procedure of obtaining these faulty configurations, they are compared. Fault collapsing is performed when different faults result in identical faulty configurations. In Table 1, the original (and the collapsed) number of faults is given, for the single and multiple LSA fault model, for the c7552, s5378, s13207 ISCAS '85 and '89 benchmark circuits [25], and for a clone from the ARM7 core [24]. Multiple LSA faults include all combinations of 2, 3 and 4 faults in the LUT inputs. As an example, there are 24 possible double faults at the inputs of a 4 input LUT: C_2^4 different pairs of nodes, where 2^2 different fault combinations can be injected. As expected, fault collapsing is more relevant for the multiple faults model.

	Single LSA		Multiple LSA	
	Total	Collapsed	Total	Collapsed
c7552	9374	7422	53566	17331
s5378	6168	4479	35100	10065
s13207	14232	10269	80180	23020
ARM7	93710	70576	624676	185473

Table 1. Original and collapsed number of LSA LUT faults

Routing configuration bits are also partially tested in this structural LUT test. In fact, LSA faults are modeled at all LUT ports (a significant sample of the CLB used logic). The impact of a routing fault on circuit functionality may manifest itself as incorrect values at LUT's inputs. However, exhaustive testing of routing faults is not carried out with LUT LSA test; hence, it is possible that, in the presence of a routing fault, a correct logic may be observed. Nevertheless, the probability of this undesired aliasing is limited when the number of observations is large.

In the example of Figure 3 there are five LUT input vectors that produce an erroneous output in the faulty LUT (corresponding to five changes in the LUT contents). If the test sequence includes one of these local input vectors, and the corresponding output is observed at the primary outputs of the core, this fault is detected by the test.

In order to test each LUT position content and reduce the aliasing probability of routing faults, a *fine grain LUT fault model* is proposed – the *Combination Stuck At* (CSA) fault model [23]. The number of faults, for this model, is identical to the possible combinations of LUT active inputs. Each fault is injected by inverting only the LUT bit position that corresponds to one input vector. The total coverage of CSA faults corresponds to the exhaustive LUT functional test for the programmed configuration. Thus, CSA is a *functionality* driven fault model, while LSA is a *line* driven fault model. It models configuration faults and also structural faults, since the LSA LUT fault model can be viewed as a multiple CSA fault model.

Table 2 compares the number of CSA faults with the number of single LSA faults for different LUT types. The column "# collapsed LSA faults" presents an approximate number of collapsed LSA faults based on the experiments reported in Section 5. This table shows that the CSA fault model leads to an increase in the fault list size, in comparison with the LSA fault list size, especially for LUTs with 4 active inputs. As these are the most used LUTs, CSA fault lists are typically more than 50 % larger than LSA fault lists. In HFE with partial reconfiguration faults are serially injected. Hence, an increase of the fault list size impacts linearly on t_{HFE}. Nevertheless, partial reconfiguration based HFE is very fast. Therefore, this increase in the fault list size is an affordable price to pay for the increase of accuracy granted by the CSA model, as it will be demonstrated in Section 5.

In order to inject a fault, the LUT content must be changed. For a given fault model, this change depends only on the LUT fault free information. LUTs with identical fault free configurations require fault injections with the same faulty LUT content. This fact allows pre-computation of the faulty LUT content, that is performed only once for a given fault model for all possible LUT contents. During the injection phase, the faulty LUT content is read from a table.

LUT type	# LSA faults	# collapsed LSA faults	# CSA faults
LUT0	2	1	1
LUT1	4	2	2
LUT2	6	4	4
LUT3	8	5	8
LUT4	10	8	16

Table 2. Number of LSA and CSA faults for different LUT types

4.2 LUT Faults vs. CLB Faults

As referred, fault injection is carried out on LUTs only. What confidence can we have in a test that, ensuring high coverage of LUTs faults, will also cover the remaining faults? In order to answer this question, one can view the proposed HFE methodology as a *sampling technique*, as LUT faults are a sample of the complete fault set. Fault sampling techniques have been widely used and documented (e.g., [1, 26]). In fact, the designer does not need to simulate the whole fault set to obtain a FC value, close to the global FC value. Therefore, for a given application, we need to evaluate how *representative* is the fault sampling that consists of taking only LUT faults to represent all CLB faults.

For this analysis, the fault list considered for the CLB consists of the LSA faults at each CLB logic element output and inputs. Faults in the selection lines of multiplexers with constant values are not considered, as they are considered as routing faults. LSA faults in routing elements are collapsed to faults at lines of logic elements.

For the FPGA components under consideration (Xilinx™ Virtex and/or Spartan FPGA), each CLB has 2 LUT and 11 additional logic elements. Table 3 shows the number and type of logic elements, the corresponding number of I/Os, the original number of faults and the collapsed number of LSA faults. Since the number of collapsed faults in a LUT with 4 inputs is 8, LUT I/O LSA faults represent only $16/(16+78) = 17\,\%$ of the CLB LSA faults.

# Cells / CLB	Cell name	# Inputs	# Outputs	# Faults	# Collapsed faults	# CLB col. faults
2	And	2	1	6	4	8
1	MUXCY	6	2	16	12	12
1	MUXF5	3	1	8	6	6
1	MUXF6	3	1	8	6	6
2	FF	5	1	12	10	20
2	XORCY	2	1	6	4	8
1	DGEN	3	1	8	8	8
1	WSGEN	4	1	10	8	10
Total number of faults					58	78

Table 3. Number of collapsed LSA faults in CLB logic elements (excluding LUT LSA faults) in Spartan and Virtex Xilinx FPGAs

However, *BIST TE must evaluate the fault coverage associated only with the circuitry used by the specific configuration*, i.e., associated with the target functionality. As shown in the next section, the use of LUT is dominant and typically fault sampling rates are over 70 %. Such significant fault sample size allows us to conclude that the sample coverage (using only LUT faults) will lead to a good estimate of the *used* CLB fault coverage, within a narrow confidence interval.

5 FAULT SIMULATION FOR FPGA CORES

The FPGA implementation of the ARM7 core [24] results, in terms of resource allocation in a Spartan 600E Xilinx FPGA, is listed in Table 4.

Table 4 presents the original and the collapsed fault list size for each logic element and the global fault list size for the circuit. For this example, using only LUT faults is equivalent to sample 70.8 % of the collapsed fault list. The huge difference from the worst-case value evaluated in the previous section is due to the majority of LUTs as processing logic elements. This is also the case for all the circuits that we have studied. Table 5 shows the fault sampling values for three additional benchmark circuits [25].

The effectiveness of the proposed HFE technique critically depends on the simulation times, t_{HFE}. Figures 4 and 5 show the fault simulation times using JTAG 200 Kb parallel III port. It is clear from these figures that HFE is advantageous over SFS after a reduced number of test vectors. After 100 000 vectors, HFE is

		Original		Collapsed	
Cell name	# Cells	# Faults / cell	# Faults	# Faults / cell	# Faults
LUT1	84	4	336	2	168
LUT1_L	15	4	60	2	30
LUT2	527	6	3162	4	2108
LUT2_D	8	6	48	4	32
LUT2_L	85	6	510	4	340
LUT3	2619	8	20952	5	13095
LUT3_D	62	8	496	5	310
LUT3_L	17	8	136	5	85
LUT4	6366	10	63660	8	50928
LUT4_D	110	10	1100	8	880
LUT4_L	325	10	3250	8	2600
MUXCY	474	16	7584	12	5688
MUXF5	824	8	6592	6	4944
MUXF6	28	8	224	6	168
XORCY	403	6	2418	4	1612
FDC	1349	8	10792	8	10792
FDCE	525	10	5250	10	5250
FDCP	30	10	300	10	300
FDP	16	8	128	8	128
FDPE	24	10	240	10	240
Total # faults		127238		99698	
# LUT faults		93710		70576	
Fault sampling %		73.6		70.8	

Table 4. Logic elements allocated for the ARM7 implementation

orders of magnitude faster than SFS. Moreover, it can be seen from Figure 5 that the new HFE tool simulates a clone from the ARM processor with 70576 faults and one million test vectors in 4800 s (one hour and twenty minutes). The flat initial part of all curves in Figure 5 is due to the fact that there is a minimum amount of time required to process the information of each fault. If the BIST session applies a limited number of vectors (TL) and runs fast (f_{HFE}), the hardware fault emulation time (t_{HFE}) is not significantly limited by the BIST time.

Circuit	c7552	s5378	s13207	ARM7
Total	13836	9592	23294	127238
# LUT faults	9374	6168	14232	93710
Total collapsed	10029	6308	14882	99698
# LUT faults collapsed	7422	4479	10269	70576
Fault sampling [%]	74,0	71,0	69,0	70.8

Table 5. Fault sampling resulting from injecting faults in LUTs

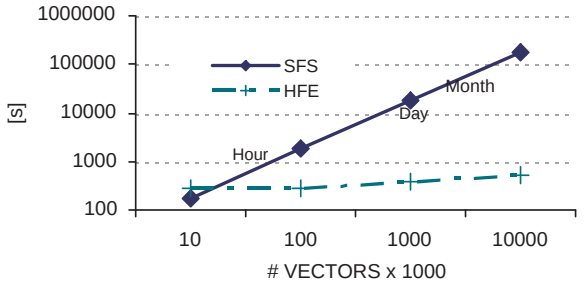

Fig. 4. HFE vs. software fault simulation time for the s13207 benchmark

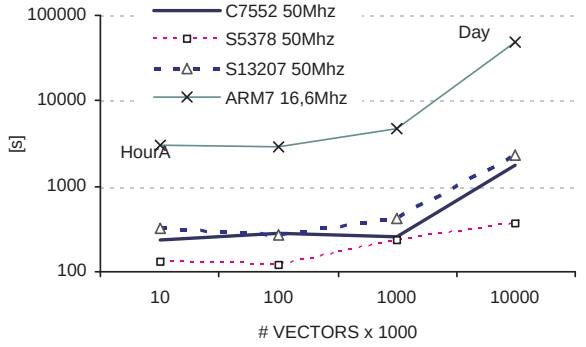

Fig. 5. Hardware fault emulation time, t_{HFE}, for different benchmark circuits

Fault coverage results can also add in deciding the test length of the BIST session, TL. As the HFE process runs fast, several experiments, with different test lengths, can be performed, providing the FC vs. the number of test vectors curve. Figure 6 shows a typical FC(TL) curve obtained using the new HFE tool and the c7552 benchmark. In order to obtain these curves, instead of using MISRs, the circuit is duplicated in the FPGA (one circuit for fault injection and one "golden" circuit) allowing fault dropping and the identification of the first vector that detects each fault. Figure 6 presents the curve obtained exercising the c7552 with an LFSR implementing a polynomial of degree 168 and with taps in positions 166, 153 and 151. The seed used was 111...1 and the fault emulation time, t_{HFE}, for 30 000 vectors, was (with $f_{HFE} = 50$ MHz) 105 seconds.

How do these results compare with the ones reported by other authors? As shown in Figure 7, adding extra hardware to enable fault injection without reconfiguration, in [9], the fault emulation time estimated for the application of 100 000 test vectors to the s13207 is 353 s for a similar number of faults (9 815). This result shows that the proposed HFE methodology, using partial reconfiguration, but no extra hardware, is a fast process. In [14], fault emulation for the same circuit with half the number of vectors takes 196 s. Without scan the fault coverage achieved is

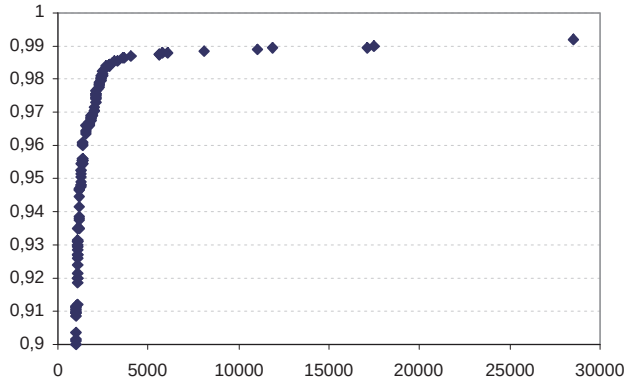

Fig. 6. Fault coverage evolution for the c7552, obtained by HFE

usually low and strongly dependant on the polynomial and seed used in the LFSR. This dependence has a huge impact in fault emulation time.

	reference [9]		partial reconf.	
	# faults	time [s]	# faults	time [s]
s5378	4603	78	4479	118
s13207	9815	353	10269	271
	100k pattern without scan path			

	reference [14]		partial reconf.	
	# faults	time [s]	# faults	time [s]
s9234	13020	148	3986	128
s13207	21256	196	10269	129
	50k pattern		100k pattern	
	with scan path			

Fig. 7. Comparing partial reconfiguration fault emulation with previous works

6 TEST PREPARATION FOR ASIC CORES

After analysis of the simulation performance and fault sample dimension of LUT faults in FPGA targeted cores, one open question is: can hardware fault emulation identify *relevant* test vectors for ASIC cores? We use the c7552 as a case study to answer this question. First, c7552 was synthesized with an AMS cell library (csx) and software fault simulation was carried out with 65 535 pseudo-random vectors. The aim of this experiment was to find out whether the HFE process (with a different structure) could lead to the identification of the test vectors that contribute to fault detection on the ASIC structure. Three HFE processes were evaluated in hardware: HFE using (1) LSA and (2) CSA fault models with the FPGA configuration obtained from *direct synthesis* of the c7552 structure, and (3) LSA fault model with the FPGA configuration obtained from *synthesis* of the c7552 AMS csx structure. This last synthesis was carried out replacing the AMS csx cells by their Register Transfer Level representation.

Figure 8 shows the results of these SFS and HFE experiments. It is clear from this figure that, with these structures and fault models, the *absolute* value of hardware fault coverage cannot be used to estimate the fault coverage of a different structure. Figure 8 also shows that the CSA fault model leads to pessimistic fault coverage results: fault coverage below 95 % after 65 535 test vectors.

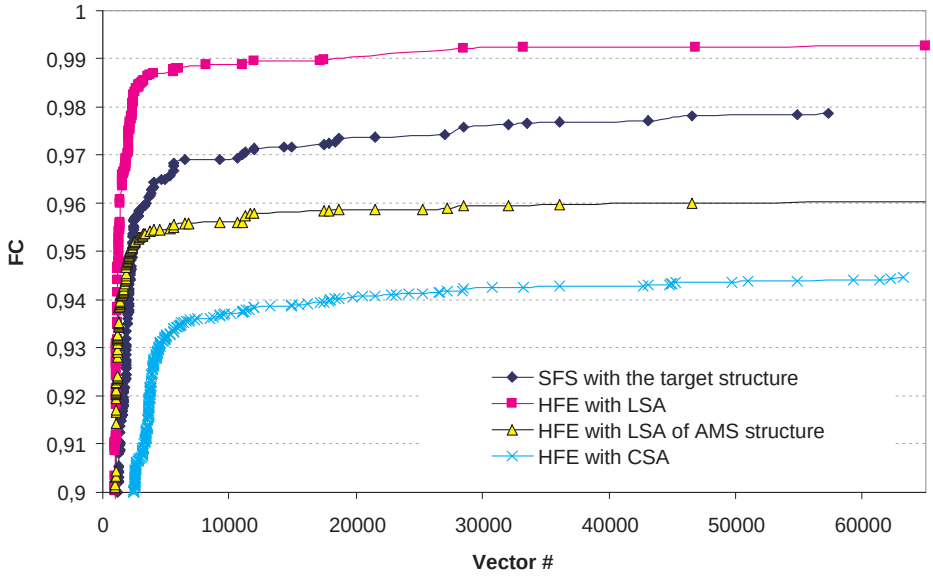

Fig. 8. Fault coverage results of the target structure and the FPGA implementations

The vectors responsible for increasing the fault coverage above 90 % were identified and compared for the different fault simulations. There are 178 vectors, out of 64 300, that are responsible for leading the fault coverage from 90 to 97.8 % in the software fault simulation. Ideally, we would like to identify these 178 vectors with HFE. Table 6 shows the number of vectors identified as *useful* by the different HFE experiments. Here, a useful test vector is one that contributes to an increase in the cumulative fault coverage value. These results show that biasing the FPGA implementation by synthesizing the target structure is rewarding. However, it still does not lead to acceptable values of vector matching (76 of 178 vectors). The fine grain model, the CSA fault model, leads to better results (127 vectors identified). Note that all of the identified test vector subsets are very limited when compared with the search space (64 300).

The low matching between test vectors identified by SFS for the target structure and the ones identified using HFE could lead to the erroneous conclusion that vector selection could not be performed in hardware. Fortunately, this is not the case since the SFS using the target structure was carried out with fault dropping. Thus, only the first vector that detects "hard faults" (that cause FC > 90 %) is identified. This

		Fault model and FPGA configuration		
		LSA	LSA (modified structure)	CSA
Matching with software fault simulation	# Useful vectors	411	423	1295
	# Identified	49	76	127
	# Not identified	129	102	51
	# Added	58	62	619

Table 6. Software and hardware vector selection matching (c7552)

means that the vectors identified in hardware as being erroneously added can, in fact, detect hard faults in the target structure.

In order to evaluate whether the vectors not previously identified as useful can detect hard faults, additional SFS was carried using the target ASIC structure. The results of these simulations are presented in Figures 9 and 10. In each simulation, the number of vectors applied is the one presented in Table 6. However, in order to evaluate the detection correlation in the different fault sets, in Figure 9, the horizontal axis is the same for all the simulations and therefore includes test vectors (most of them) that were not applied.

Figure 9 shows clearly that the vectors identified in hardware are far more useful than the matching with the first detection (Table 6) could suggest. In fact, the fault coverage gain achieved with these vectors is much higher than the matching shown in Table 6. In fact, LSA-based HFE leads to the detection of 60 % of the hard faults with the standard FPGA configuration. The detection value increases to 80 % when using the AMS csx structure, more similar to the ASIC structure. Using the CSA fault model and the standard FPGA configuration, 91 % of the hard faults are detected.

7 CONCLUSIONS

Test quality assessment, namely BIST test effectiveness evaluation for stand-alone or embedded sequential cores is a costly process. In this work, a novel HFE methodology and tool is proposed, that efficiently, using partial reconfiguration, ascertain (or not) the BIST solution under scrutiny. The proposed HFE methodology has been compared to a commercial software fault simulation (SFS) tool and the superiority has been demonstrated on ISCAS '85 and '89 benchmark circuits. The new HFE tool simulates a clone from the ARM7 processor with 70576 faults and one million test vectors in less than two hours, operating the FPGA emulator at 16.6 MHz. The LUT extraction and partial reconfiguration processes were detailed for Xilinx^{TM} Virtex and/or Spartan FPGA components.

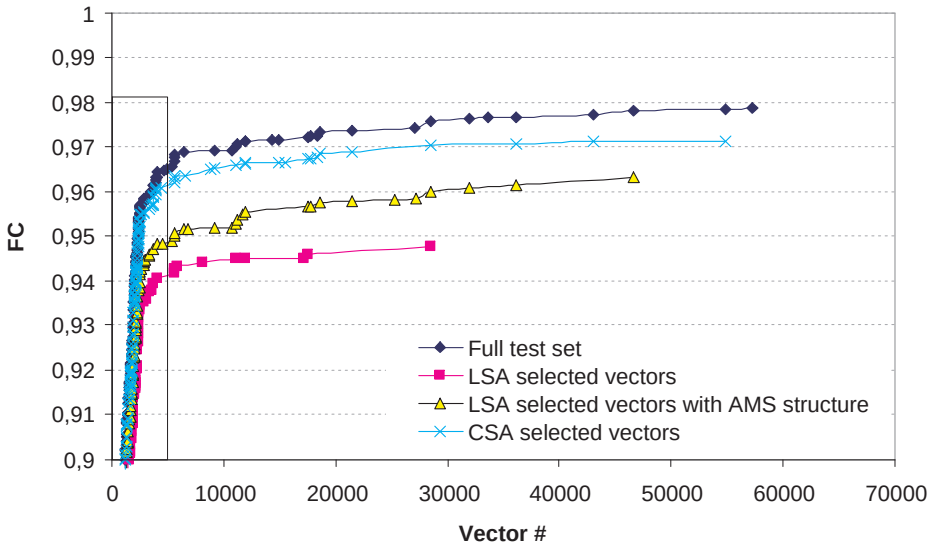

Fig. 9. SFS using the target structure with different test sets (c7552)

A new, efficient and flexible set of LUT fault models was implemented. The faulty LUT configurations that correspond to each fault are pre-computed only once, prior to fault emulation, and loaded in the beginning of the HFE process. This approach is more efficient and adds flexibility to the fault model.

The CLB fault list was detailed and the use of the LSA fault model at LUTs terminals was analyzed as a fault sampling technique. Starting from a pessimistic 17% fault sampling value if all CLB resources were in use, it has been shown that, due to the relevance of the LUTs on the implementation of the FPGA logic, the fault sampling is over 70% in practical cases.

Test preparation in FPGA for ASIC cores was analyzed from two different points of view: the capability of HFE to identify the first vector that detects each hard fault and the fault coverage achieved in an ASIC structure using vectors selected by HFE. While vector matching was reduced, the vectors selected using HFE and the CSA fault model lead to very high LSA fault coverage in the ASIC structure: 97.14 % (1 295 selected vectors) of a maximum of 97.87 % (65 535 vectors), for the c7552 benchmark.

Acknowledgement

This work has been partially funded by FCT (Portugal), under Project POCTI/ 41788/ESE/2001.

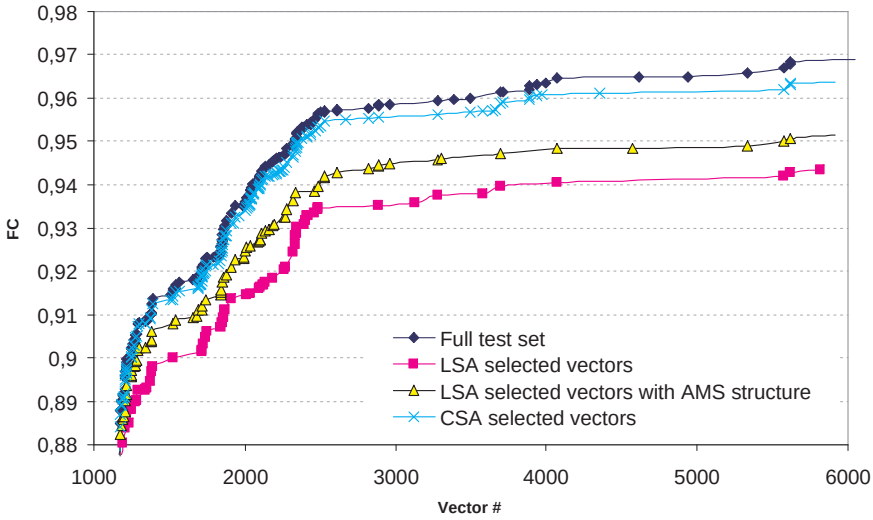

Fig. 10. SFS using the target structure with different test sets (zoom of the first 6 000 vectors fault coverage) (c7552)

Glossary

ASIC – Application Specific Integrated Circuit
BIST – Built-In Self-Test
CLB – Configurable Logic Block
CSA – Combination Stuck-At
DfT – Design for Testability
FC – Fault Coverage
FS – Fault Simulation
FPGA – Field Programmable Gate Arrays
HFE – Hardware Fault Emulation
LFSR – Linear Feedback Shift Register
LSA – Line Stuck-At
LUT – Look-Up Table
MISR – Multi-Input Shift Register
PR – Pseudo-random
SFS – Software Fault Simulation
TE – Test Effectiveness

TL – Test Length

TO – Test Overhead

TP – Test Power

TPG – Test Pattern Generator

REFERENCES

[1] Bushnel, M. L.—Agrawal, V. D.: Essentials of Electronic Testing for Digital Memory and Mixed-Signal VLSI Circuits. Kluwer Academic Pubs., 2000.

[2] Stroud, Ch. E.: A Designer's Guide to Built-In Self Test. Kluwer Academic Pubs., ISBN 1-4020-7050-0, 2002.

[3] Sousa, J. J. T.—Gonçalves, F. M.—Teixeira, J. P.—Marzocca, C.—Corsi, F.—Williams, T. W.: Defect Level Evaluation in an IC Design Environment. IEEE Trans. on CAD, Vol. 15, 1996, No. 10, pp. 1286–1293.

[4] Niermann, T. M.—Cheng, W. T.—Patel, J. H.: PROFS: A Fast, Memory-Efficient Sequential Circuit Fault Simulator. IEEE Trans. Computer-Aided Design, 1992, pp. 198–207.

[5] Rudnick, E. M.—Patel, J. H.: Overcoming the Serial Logic Simulation Bottleneck in Parallel Fault Simulation. Proc. of the IEEE International Test Conference (ITC), 1997, pp. 495–501.

[6] Gonçalves, F. M.—Santos, M. B.—Teixeira, I. C.—Teixeira, J. P.: Design and Test of a Certifiable ASIC for Safety-Critical Gas Burners Control System. Journal of Electronic Testing, Theory and Application (JETTA), Vol. 18, No. 3, Kluwer Academic Publishers, June 2002, pp. 285-294.

[7] Wieler, R. W.—Zhang, Z.—McLeod, R. D.: Simulating Static and Dynamic Faults in BIST Structures with a FPGA Based Emulator. Proc. of IEEE Int. Workshop of Field-Programmable Logic and Application, pp. 240–250, 1994.

[8] Cheng, K.—Huang, S.—Dai, W.: Fault Emulation: A New Methodology for Fault Grading. IEEE Trans. On Computer-Aided Design of Integrated Circuits and Systems, Vol. 18, 1999, No. 10, pp. 1487–1495.

[9] Hwang, Sh.-A.—Hong, J.-H.—Wu, Ch.-W.: Sequential Circuit Fault Simulation Using Logic Emulation. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. 17, 1998, No. 8, pp. 724—736.

[10] Civera, P.—Macchiarulo, L.—Rebaudengo, M.—Sonza Reorda, M.—Violante, M.: An FPGA-Based Approach for Speeding-Up Fault Injection Campaigns on Safety-Critical Circuits. IEEE Journal of Electronic Testing Theory and Applications, Vol. 18, 2002, No. 3, pp. 261–271.

[11] Santos, M. B.—Braga, J.—Teixeira, I. M.—Teixeira, J. P.: Dynamic Fault Injection Optimization for FPGA-Based Hardware Fault Simulation. Proc. of the Design and Diagnostics of Electronic Circuits and Systems Workshop (DDECS), April, 2002, pp. 370–373.

[12] Abramovici, M.—Menon, P.: Fault Simulation on Reconfigurable Hardware. IEEE Symposium on FPGAs for Custom Computing Machines, 1997, pp. 182–190.

[13] Abramovici, M.—Menon, P. R.—Miller, D. T.: Critical Path Tracing: An Alternative to Fault Simulation. IEEE Design Automation Conference, 1984, pp. 468–474.

[14] Burgun, L.—Reblewski, F.—Fenelon, G.—Barbier, J.—Lepape, O.: Serial Fault Simulation. Proc. Design Automation Conference, 1996, pp. 801–806.

[15] Antoni, L.—Leveugle, R.—Fehér, B.: Using Run-Time Reconfiguration for Fault Injection in Hardware Prototypes. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, October 2000, pp. 405–413.

[16] Antoni, L.—Leveugle, R.—Fehér, B.: Using Run-Time Reconfiguration for Fault Injection Applications. IEEE Instrumentation and Measurement Technology Conference, Vol. 3, May 2001, pp. 1773–1777.

[17] Antoni, L.—Leveugle, R.—Fehér, B.: Using Run-Time Reconfiguration for Fault Injection in Hardware Prototypes. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2002, pp. 245–253.

[18] Guccione, S.—Levi, D.—Sundararajan, P.: Jbits: A Java-based Interface for Reconfigurable Computing. Proc. of the 2nd Military and Aerospace Applications of Programmable Devices and Technologies Conf. (MAPLD), 1999, pp. 27.

[19] Lechner, E.—Guccione, S.: The Java Environment for Reconfigurable Computing. Proc. of the 7th Int. Workshop on Field-Programmable Logic and Applications (FPL), Lecture Notes in Comp. Science 1304, September 1997, pp. 284–293.

[20] Sundararajan, P.—Guccione, S.—Levi, D.: XHWIF: A Portable Hardware Interface for Reconfigurable Computing. Proc. of Reconfigurable Technology: FPGAs and Reconfigurable Processors for Computing and Communications, SPIE 4525, August 2001, pp. 97–102.

[21] Xilinx Inc., Virtex-E 1.8V Field Programmable Gate Arrays. Xilinx DS022, 2001.

[22] Xilinx Inc., Virtex Series Configuration Architecture User Guide. Application Note: Virtex Series, XAPP151 (v1.5), September 27, 2000.

[23] Parreira, A.—Teixeira, J. P.—Santos, M. B.: A Novel Approach to FPGA-based Hardware Fault Modeling and Simulation. Proc. of the Design and Diagnostics of Electronic Circuits and System Workshop, April, 2003, pp. 17–24.

[24] Sheng Yu Shen, nnARM core a ARM-7 clone, :`http://www.opencores.org`.

[25] Brglez, F.—Bryan, D.—Kominski, K.: Combinational Profiles of Sequential Benchmark Circuits. Proc. Int. Symp. on Circuits and Systems (ISCAS), 1989, pp. 1229–34.

[26] McNamer, M. G.—Roy, S. C.—Nagle, H. T.: Statistical Fault Sampling. IEEE Trans. on Industrial Electronics, Vol. 36, 1989, No. 2, pp. 141–150.

[27] Parreira, A.—Teixeira, J. P.—Santos, M. B.: FPGAs BIST Evaluation. Proc. of the 14th Int. Workshop on Field-Programmable Logic and Applications (FPL), August 2004, pp. 333–343.

Abílio Mendes Parreira has a large experience as a programmer and analyst. He has got his electrical engineer degree by IST (Instituto Superior Técnico) from the Technical University of Lisbon, where he is currently finishing his M.Sc. thesis. He is a researcher at INESC-ID (Instituto de Engenharia de Sistemas e Computadores – Investigação e Desenvolvimento), in ProSyS – the Programmable Systems Laboratory. His interests include design and test of digital systems, fault modelling and simulation, Built In Self Test (BIST), defect-oriented test and the development of EDA (Electronic Design Automation) tools using reconfigurable hardware.

Marcelino Bicho dos Santos has got his electrical engineer degree, M.Sc. degree and his Ph.D. in electrical and computer engineering by IST (Instituto Superior Técnico) from the Technical University of Lisbon. He is currently an Assistant Professor at IST. He holds a Senior researcher position at INESC-ID (Instituto de Engenharia de Sistemas e Computadores – Investigação e Desenvolvimento) in the Programmable Systems Laboratory. His scientific and technical interests include design and test of digital, analog and mixed-signal circuits and systems. His previous research works addresses test preparation at register transfer level, fault modelling and simulation, Built In Self Test (BIST) and defect-oriented test.

Joao Paulo Teixeira has got his electrical engineer degree (Telecommunications and Electronics) and his Ph.D. in Applied Electronics by IST (Instituto Superior Técnico) from the Technical University of Lisbon. He is currently an Associate Professor at IST. He holds a Senior researcher position at INESC-ID (Instituto de Engenharia de Sistemas e Computadores – Investigação e Desenvolvimento), where he is Head of ProSyS – the Programmable Systems Laboratory. His scientific and technical interests include the analysis, specification, design, production and test of hardware/software systems, which use microelectronics as supporting technologies. Emphasis is given to the design and test of digital, analog and mixed-signal circuits and systems, implemented in CMOS or CMOS-compatible nanometer technologies, and to the development of EDA (Electronic Design Automation) tools, especially CAT (Computer-Aided Testing) tools.